



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение высшего  
образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

---

Институт информационных технологий (ИТ)  
Кафедра математического обеспечения и стандартизации информационных  
технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №7\_1**  
**по дисциплине**  
**«Структуры и алгоритмы обработки данных»**

Тема. Балансировка дерева поиска

Выполнил студент группы ИКБО-42-23

Голев С.С.

Принял ассистент

Муравьёва Е.А.

Москва 2024

## ОГЛАВЛЕНИЕ

<b>ЦЕЛЬ РАБОТЫ .....</b>	<b>3</b>
<b>УСЛОВИЯ ЗАДАЧ .....</b>	<b>4</b>
<b>1. ОТЧЁТ ПО ЗАДАНИЮ 1.....</b>	<b>6</b>
<b>2. ОТЧЁТ ПО ЗАДАНИЮ 2.....</b>	<b>7</b>
<b>2.1. Алгоритм работы функций .....</b>	<b>7</b>
<b>2.2. Код используемый в программе .....</b>	<b>8</b>
<b>2.3. Результаты тестирования .....</b>	<b>11</b>
<b>3. ОТЧЁТ ПО ЗАДАНИЮ 3.....</b>	<b>13</b>
<b>3.1. Алгоритм работы функций .....</b>	<b>13</b>
<b>3.2. Код используемый в программе .....</b>	<b>14</b>
<b>3.3. Результаты тестирования .....</b>	<b>18</b>
<b>4.ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ .....</b>	<b>20</b>

## **ЦЕЛЬ РАБОТЫ**

Получение умений и навыков разработки и реализации операций над структурой данных бинарное дерево.

## УСЛОВИЯ ЗАДАЧ

### **Задание 1:**

Вопросы.

### **Задание 2:**

Разработать программу бинарного дерева поиска и выполнить операции в соответствии с требованиями варианта.

Методы, которые должны быть реализованы в независимости от варианта:

- включение элемента в дерево;
- поиск ключа в дереве;
- удаление ключа из дерева;
- отображение дерева.

Целое число	<ol style="list-style-type: none"><li>1. Определить среднее арифметическое всех узлов дерева, используя алгоритм обхода в «ширину».</li><li>2. Определить количество узлов в дереве.</li><li>3. Удалить максимальный элемент дерева (Считать, что такой элемент один)</li></ol>
-------------	---

### **Задание 3:**

Разработать приложение, которое использует сбалансированное дерево поиска (СДП), и выполнить операции в соответствии с требованиями варианта.

Методы, которые должны быть реализованы в независимости от варианта:

- включение элемента в дерево;
- удаление ключа из дерева;
- поиск ключа в дереве с возвратом записи из файла;
- вывод дерева в форме дерева (с отображением структуры дерева).

В-дерево	Целое число	<ol style="list-style-type: none"> <li>1. Используя рекурсивный алгоритм определить количество уровней в дереве.</li> <li>2. Вернуть узел с максимальным значением, обходя дерево в ширину.</li> <li>3. Определить разницу между максимальным и минимальным значениями</li> </ol>
----------	-------------	---

## 1. ОТЧЁТ ПО ЗАДАНИЮ 1

1. Что определяет степень дерева?

Степень дерева определяется как максимальное количество дочерних узлов, которые могут иметь узлы данного дерева.

2. Какова степень сильноветвящегося дерева?

Это максимальное количество дочерних узлов, которые могут иметь узлы данного дерева.

3. Что определяет путь в дереве?

Узлы, ребра.

4. Как рассчитать длину пути в дереве?

Если путь проходит через  $n$  узлов, длина пути будет равна  $n - 1$ , так как длина пути определяется количеством ребер, соединяющих узлы.

5. Какова степень бинарного дерева?

2

6. Может ли дерево быть пустым?

да

7. Дайте определение бинарного дерева?

Структура данных, в которой каждый узел имеет не более двух дочерних узлов.

8. Дайте определение алгоритму обхода.

Метод, используемый для посещения и обработки всех узлов в структуре данных.

9. Что такое высота дерева? Приведите рекуррентную зависимость для вычисления высоты дерева

Максимальное количество ребер на самом длинном пути от корня дерева до лист.

10. Какая структура используется в алгоритме обхода дерева методом в «ширину»

Очередь.

11. Какая структура используется в не рекурсивном обходе дерева методом в «глубину»

Стек.

12. В каком порядке будет проходиться бинарное дерево, если алгоритм обхода в ширину будет запоминать узлы не в очереди, а в стеке

Сначала дочерние узлы, потом соседи.

## 2. ОТЧЁТ ПО ЗАДАНИЮ 2

### 2.1. Алгоритм работы функций

`insert` – происходит сравнение с узлами и спуск, пока не будет свободного места

`search` – происходит сравнение с узлами и спуск, пока значение поиска и узла не совпадут

`remove` – строится второе дерево, которое копирует изначальное дерево без одного элемента

`display_line` – происходит вывод дерева в инфиксной форме (левое поддерево -> корень -> правое поддерево)

`display` – происходит вывод дерева в инфиксной форме (левое поддерево -> корень -> правое поддерево)

`average` – происходит обход в инфиксной форме и запоминаются максимальное и минимальное значения

`node_count` – происходит обход в инфиксной форме и подсчёт количества узлов

`max_el` – переход к самому левому элементу дерева и возвращение этого элемента

## 2.2. Код используемый в программе

Опишем функции, используемые в программе для решения задач.

```
Node* insert(Node* node, int value)
{
    if (node == nullptr)
    {
        return new Node(value);
    }
    if (search(node,value))
    {
        std::cout << "Повторная вставка" << std::endl;
        return root;
    }
    if (value < node->data)
    {
        node->left = insert(node->left, value);
    }
    else {
        node->right = insert(node->right, value);
    }
    return node;
}
```

*Рисунок 2.1 – Функция вставки*

```
Node* search(Node* node, int value)
{
    if (node == nullptr || node->data == value)
    {
        return node;
    }
    if (value < node->data) {
        return search(node->left, value);
    }
    else {
        return search(node->right, value);
    }
}
```

*Рисунок 2.2 – Функция поиска*



```

void remove_pr(Node* node, int value)
{
    if (node != nullptr)
    {
        if (node->data != value)
            insert(node->data);
        remove_pr(node->left, value);
        remove_pr(node->right, value);
    }
}

```

*Рисунок 2.3 – Функция удаления элемента*

```

void display_line(Node* root)
{
    if (root != nullptr)
    {
        display_line(root->left);
        std::cout << root->data << " ";
        display_line(root->right);
    }
}

```

*Рисунок 2.4 – Функция вывода*

```

void display(Node* node, int space = 0, int level = 0)
{
    int count = 5;
    if (node == nullptr) return;

    space += count;

    display(node->right, space, level + 1);

    std::cout << std::endl;
    for (int i = count; i < space; i++) {
        std::cout << " ";
    }
    std::cout << node->data << "\n";

    display(node->left, space, level + 1);
}

```

*Рисунок 2.5 – Функция вывода в виде графа*

```

void average(Node* node)
{
    globSum = 0;
    globSum += node->data;
    globSize = 1;
    traversal(node);
}

void traversal(Node* node)
{
    if (node == nullptr) return;
    if (node->left != nullptr)
    {
        globSum += (node->left->data);
        globSize++;
    }
    if (node->right != nullptr)
    {
        globSum += (node->right->data);
        globSize++;
    }
    traversal(node->left);
    traversal(node->right);
}

```

*Рисунок 2.6 – Функции поиска среднего арифметического*

```

int max_el(Node* node)
{
    if (node->right != nullptr)
        return max_el(node->right);
    else
        return node->data;
}

```

*Рисунок 2.7 – Функции поиска максимального элемента*

## 2.3. Результаты тестирования

Протестируем функции, используемые в втором задании.

```
Введите число для вставки: 4

Для продолжения нажмите любую клавишу . . .

Выберите команду:
1. Вставка;
2. Поиск;
3. Удаление;
4. Вывод (инфиксная форма);
5. Вывод (граф);
6. Нахождение среднего арифметического;
7. Количество узлов;
8. Максимальный элемент дерева
Команда: 5

      7
     / \
    5   4
   / \
  3   4
```

Рисунок 2.8 – Тестирование функции вставки

```
Введите число для удаления: 4

Для продолжения нажмите любую клавишу . . .

Выберите команду:
1. Вставка;
2. Поиск;
3. Удаление;
4. Вывод (инфиксная форма);
5. Вывод (граф);
6. Нахождение среднего арифметического;
7. Количество узлов;
8. Максимальный элемент дерева
Команда:
5

      7
     / \
    5   4
   / \
  3   4
```

Рисунок 2.9 – Тестирование функции удаления

```
Введите число для поиска: 7
Узел с значением 7 найден.
```

Рисунок 2.9 – Тестирование функции поиска

```
Команда: 6
Среднее арифметическое дерева: 5
```

Рисунок 2.10 – Тестирование функции поиска среднего арифметического

```
Количество узлов дерева: 3
```

Рисунок 2.11 – Тестирование функции подсчёта узлов

```
Максимальный элемент удалён
Для продолжения нажмите любую клавишу . . .
Выберите команду:
    1. Вставка;
    2. Поиск;
    3. Удаление;
    4. Вывод (инфиксная форма);
    5. Вывод (граф);
    6. Нахождение среднего арифметического;
    7. Количество узлов;
    8. Максимальный элемент дерева
Команда: 5
5
3
```

Рисунок 2.12 – Тестирование функции удаления максимального узла

### 3. ОТЧЁТ ПО ЗАДАНИЮ 3

#### 3.1. Алгоритм работы функций

`insert` – поиск узла со свободным местом.

`insert_non_full` – вставка в неполный узел.

`split_child` – создание нового узла из среднего элемента, и добавление в дочерние узлы две части предыдущего узла.

`search` – обход узла, проверка на совпадение, последующий спуск, пока не найдётся совпадение.

`remove` – обход узла, проверка на совпадение, последующий спуск, пока не найдётся совпадение, после находа удаление элемента в узле.

`display` – обход узла вывод его элементов, переход к дочерним узлам.

`find_levels` – обход всех узлов и сравнение и поиск самого глубокого, посредством сравнения с имеющейся высотой.

`max_node` – обход всех узлов, пока не найдётся самый правый, который хранит максимальные значения.

`max_min` – обход всех узлов в ширину, и сравнение с имеющимися элементами, для нахождения максимального и минимального элемента

### 3.2. Код используемый в программе

```
void insert(int key)
{
    if (search(root, key))
    {
        std::cout << "Повторная вставка" << std::endl;
        return;
    }
    if (root == nullptr)
    {
        root = new BTreeNode(t, true);
        root->keys.push_back(key);
    }
    else
    {
        if (root->keys.size() == 2 * t - 1)
        {
            BTreeNode* newRoot = new BTreeNode(t, false);
            newRoot->children.push_back(root);
            split_child(newRoot, 0, root);
            root = newRoot;
        }
        insert_non_full(root, key);
    }
}
```

*Рисунок 3.1 – Функция вставки*

```

void insert_non_full(BTreeNode* node, int key)
{
    int i = node->keys.size() - 1;

    if (node->isLeaf)
    {
        node->keys.push_back(0);
        while (i >= 0 && key < node->keys[i])
        {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
    }
    else
    {
        while (i >= 0 && key < node->keys[i])
            i--;

        if (node->children[i + 1]->keys.size() == 2 * t - 1)
        {
            split_child(node, i + 1, node->children[i + 1]);
            if (key > node->keys[i + 1])
                i++;
        }
        insert_non_full(node->children[i + 1], key);
    }
}

```

*Рисунок 3.2 – Функция вставки в неполный узел*

```

void split_child(BTreeNode* parent, int i, BTreeNode* child)
{
    BTreeNode* newNode = new BTreeNode(t, child->isLeaf);

    for (int j = 0; j < t - 1; j++)
        newNode->keys.push_back(child->keys[j + t]);

    if (!child->isLeaf)
        for (int j = 0; j < t; j++)
            newNode->children.push_back(child->children[j + t]);

    parent->children.insert(parent->children.begin() + i + 1, newNode);
    parent->keys.insert(parent->keys.begin() + i, child->keys[t - 1]);

    child->keys.resize(t - 1);
    child->children.resize(t);
}

```

*Рисунок 3.3 – Функция разбиения переполненного узла*

```

BTreeNode* search(BTreeNode* node, int key)
{
    if (node == nullptr) return nullptr;

    int i = 0;
    while (i < node->keys.size() && key > node->keys[i])
        i++;

    if (i < node->keys.size() && key == node->keys[i])
        return node;

    if (node->isLeaf)
        return nullptr;

    return search(node->children[i], key);
}

```

*Рисунок 3.4 – Функция поиска элемента*

```

void display(BTreeNode* node, int depth)
{
    int i;
    std::cout << std::setw(depth * 4) << " " << "+--";
    std::cout << '[';
    for (i = 0; i < node->keys.size(); i++)
    {
        std::cout << node->keys[i];
        if (i != node->keys.size() - 1)
            std::cout << '|';
    }
    std::cout << "] ";
    std::cout << std::endl;
    for (i = 0; i < node->children.size(); i++)
    {
        if (!node->isLeaf)
        {
            display(node->children[i], depth + 1);
        }
    }
}

```

*Рисунок 3.5 – Функция вывода дерева*



```

BTreeNode* remove(BTreeNode* node, int key)
{
    if (node == nullptr) return nullptr;

    int i = 0;
    while (i < node->keys.size() && key > node->keys[i])
        i++;

    if (i < node->keys.size() && key == node->keys[i])
        node->keys.erase(node->keys.begin() + i);

    if (node->isLeaf)
        return nullptr;

    return remove(node->children[i], key);
}

```

*Рисунок 3.6 – Функция удаления элемента*

```

int find_levels(BTreeNode* node)
{
    if (node == nullptr)
        return 0;
    int Height = 0;
    for (int i = 0; i < node->children.size(); i++)
    {
        Height = std::max(Height, find_levels(node->children[i]));
    }
    return Height + 1;
}

```

*Рисунок 3.7 – Функция вычисления высоты*

```

BTreeNode* max_node(BTreeNode* node)
{
    if (!(node->isLeaf))
    {
        for (int i = 0; i < node->children.size(); i++)
        {
            if (i + 1 == node->children.size())
                max_node(node->children[i]);
        }
    }
    else
    {
        return node;
    }
}

```

*Рисунок 3.8 – Функция вычисления узла с максимальным значением*

```

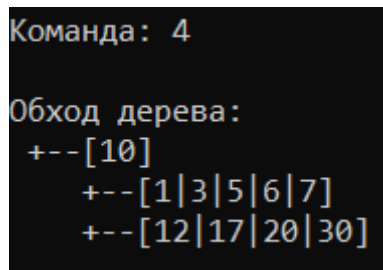
void max_min(BTreeNode* node)
{
    for (int i = 0; i < node->keys.size(); i++)
    {
        maxEl = std::max(maxEl, node->keys[i]);
        minEl = std::min(minEl, node->keys[i]);
    }
    if (!(node->isLeaf))
        for (int i = 0; i < node->children.size(); i++)
            max_min(node->children[i]);
}

```

*Рисунок 3.9* – Функция вычисления разницы максимального и минимального элементов

### 3.3. Результаты тестирования

Протестируем функции, используемые в третьем задании.



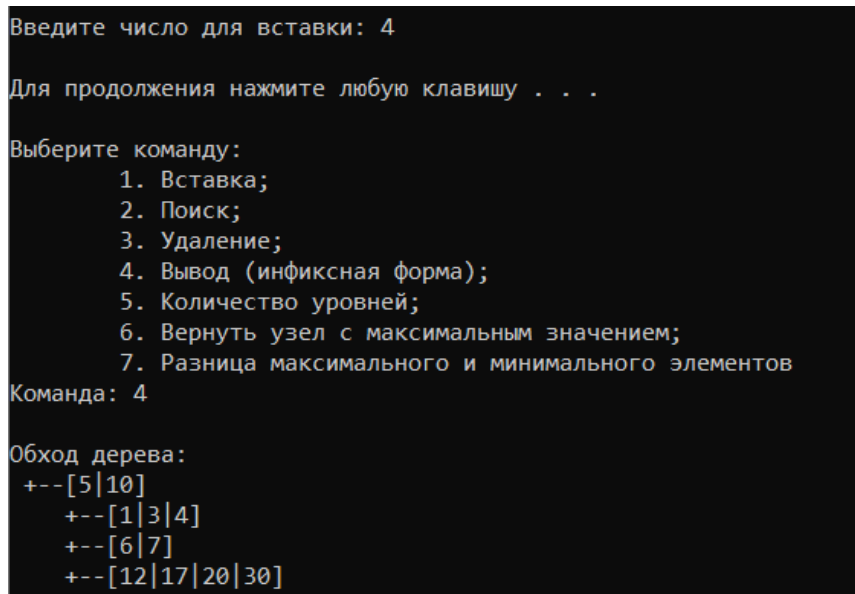
```

Команда: 4

Обход дерева:
+--[10]
+--[1|3|5|6|7]
+--[12|17|20|30]

```

*Рисунок 3.10* – Тестирование функции вывода



```

Введите число для вставки: 4

Для продолжения нажмите любую клавишу . . .

Выберите команду:
1. Вставка;
2. Поиск;
3. Удаление;
4. Вывод (инфиксная форма);
5. Количество уровней;
6. Вернуть узел с максимальным значением;
7. Разница максимального и минимального элементов
Команда: 4

Обход дерева:
+--[5|10]
+--[1|3|4]
+--[6|7]
+--[12|17|20|30]

```

*Рисунок 3.11* – Тестирование функции вставки

```
Введите число для удаления: 4
Для продолжения нажмите любую клавишу . . .
Выберите команду:
    1. Вставка;
    2. Поиск;
    3. Удаление;
    4. Вывод (инфиксная форма);
    5. Количество уровней;
    6. Вернуть узел с максимальным значением;
    7. Разница максимального и минимального элементов
Команда: 4
Обход дерева:
+--[5|10]
  +--[1|3]
  +--[6|7]
    +--[12|17|20|30]
```

Рисунок 3.12 – Тестирование функции удаления

```
Команда: 5
Количество уровней дерева: 2
```

Рисунок 3.13 – Тестирование функции вычисления высоты дерева

```
Команда: 6
Узел с максимальным значением возвращён.
```

Рисунок 3.14 – Тестирование функции возвращения узла с максимальными элементами

```
Команда: 7
Разница максимального и минимального элемента: [29]
```

Рисунок 3.15 – Тестирование функции вычисления разницы между максимальным и минимальным элементами

#### **4.ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ**

1. Лекции по Структуры и алгоритмы обработки данных / Рысин М. Л.  
Москва, МИРЭА — Российский технологический университет.
2. Материалы по дисциплине Структуры и алгоритмы обработки данных /  
Скворцова Л. А. Москва, МИРЭА — Российский технологический университет.